

Data Integration Flows for Business Intelligence

Umeshwar Dayal
HP Labs
Palo Alto, Ca, USA

umeshwar.dayal@hp.com

Malu Castellanos
HP Labs
Palo Alto, Ca, USA

malu.castellanos@hp.com

Alkis Simitsis
HP Labs
Palo Alto, Ca, USA

alkis@hp.com

Kevin Wilkinson
HP Labs
Palo Alto, Ca, USA

kevin.wilkinson@hp.com

ABSTRACT

Business Intelligence (BI) refers to technologies, tools, and practices for collecting, integrating, analyzing, and presenting large volumes of information to enable better decision making. Today's BI architecture typically consists of a data warehouse (or one or more data marts), which consolidates data from several operational databases, and serves a variety of front-end querying, reporting, and analytic tools. The back-end of the architecture is a data integration pipeline for populating the data warehouse by extracting data from distributed and usually heterogeneous operational sources; cleansing, integrating and transforming the data; and loading it into the data warehouse. Since BI systems have been used primarily for off-line, strategic decision making, the traditional data integration pipeline is a one-way, batch process, usually implemented by extract-transform-load (ETL) tools. The design and implementation of the ETL pipeline is largely a labor-intensive activity, and typically consumes a large fraction of the effort in data warehousing projects. Increasingly, as enterprises become more automated, data-driven, and real-time, the BI architecture is evolving to support operational decision making. This imposes additional requirements and tradeoffs, resulting in even more complexity in the design of data integration flows. These include reducing the latency so that near real-time data can be delivered to the data warehouse, extracting information from a wider variety of data sources, extending the rigidly serial ETL pipeline to more general data flows, and considering alternative physical implementations. We describe the requirements for data integration flows in this next generation of operational BI system, the limitations of current technologies, the research challenges in meeting these requirements, and a framework for addressing these challenges. The goal is to facilitate the design and implementation of optimal flows to meet business requirements.

Keywords

Data Warehousing, Business Intelligence, ETL, Data Integration.

1. INTRODUCTION

Business Intelligence (BI) is a collection of data warehousing, data mining, analytics, reporting and visualization technologies,

tools, and practices to collect, integrate, cleanse, and mine enterprise information for decision making. Today's BI architecture was designed for strategic decision making, where a small number of expert users analyze historical data to prepare reports or build models, and decision making cycles last weeks or months. This architecture may be viewed as an information supply chain (Figure 1). Data from distributed, often heterogeneous, sources such as online transaction processing (OLTP) systems is periodically extracted, cleansed, integrated, transformed, and loaded into a data warehouse (DW), which in turn is queried by analytic applications [4]. (Sometimes, organizations choose to construct Data Marts, each of which contains information on some subset of the subject areas represented in the DW.) Traditionally, the back-end of the information supply chain is a one-way, batch process (a data pipeline) usually implemented by home-grown code or extract-transform-load (ETL) tools, such as Informatica's PowerCenter, DataStage, Ab Initio, Oracle's Warehouse Builder, and so on.

Historically, ETL design and implementation was considered a supporting task for the data warehouse, and was largely ignored by the research community. In fact, a seminal book on data warehousing does not explicitly mention ETL at all, although the concepts of extraction, transformation and loading are described [18]. Perhaps ETL received so little attention because, conceptually, it appeared to be a relatively simple task of data transfer and integration [33]. Yet, it remains an expensive, labor-intensive, largely manual task. In fact, in a typical data warehouse project, ETL can consume a large fraction of the effort (70 percent by some estimates).

The focus for ETL has been on correct functionality and adequate performance; i.e., the functional mappings from data sources to warehouse must be correct and the ETL mappings must complete within a certain time window. However, an ETL project that focuses just on functionality and performance misses other business objectives that, while harder to quantify, are important to success. As one ETL practitioner told us, "If I wanted better performance I buy better hardware; unfortunately, I cannot buy a more maintainable or a more reliable system."

In addition, as enterprises become more automated, real-time, and data-driven, the industry is evolving toward BI systems that support online, *operational* decision making at all levels in the enterprise [12, 36]. There is a growing realization that BI must be integrated into the business operations of the enterprise to enable the many knowledge workers engaged in business processes to make better and timelier decisions. High quality information must be delivered in near real-time to analytic applications that are integrated into the enterprise's business processes. For example, an on-line retailer would like to analyze

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EDBT'09, March 24-26, 2009, Saint Petersburg, Russia.

Copyright 2009 ACM 978-1-60558-422-5/09/0003 ...\$5.00.

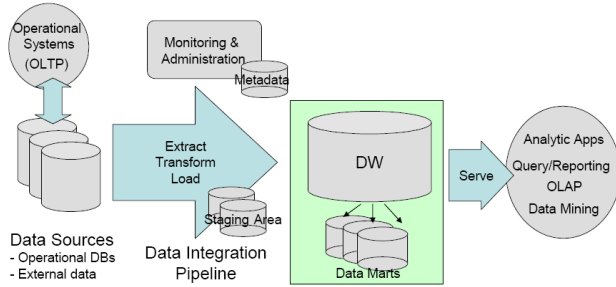


Figure 1. Traditional business intelligence architecture

a user's real-time click stream data and up-to-the-minute inventory to offer dynamically priced product bundles. A bank would like to detect and react in real-time to fraudulent transactions. A logistics provider would like to dynamically reconfigure shipping routes in response to weather conditions.

Operational BI imposes several new requirements on the architecture, and in particular on the back-end data integration processes. These include: handling a much larger number and diversity of data sources and data types (unstructured and semi-structured enterprise content, external data feeds, sensor and other forms of streaming data), low latency requirements to support on-line decision making, fast refresh cycles, more complex analytic and reporting tools, a larger number of data mart connections, 24x7 availability, and so on. In the evolving architecture for operational BI, ETL processes are no longer a one-way, batch pipeline, but they become more general data flows, where for instance events from the sources are streamed through transformation operations towards the data warehouse, and cleansed data may flow back to the operational databases (Figure 2).

With these increasing demands on data warehouses, ETL design has become even more complex. Consequently, we feel it is time to take a fresh, comprehensive look at the problem of data integration flow design and implementation.

Our objective in this paper is to discuss research problems and a promising framework for addressing them, not to describe solutions. In Section 2, we survey the requirements we see for next generation data integration flows, and the challenges they pose. In Section 3, we describe a layered methodology that allows us to capture the requirements starting at the business level, and progressing to an optimized, executable implementation. Section 4 describes a set of metrics for data integration flow design and implementation, and illustrates tradeoffs among these metrics. Section 5 discusses problems and techniques in optimizing flows. Section 6 discusses issues in incorporating new data types, i.e., unstructured and semi-structured data.

2. NEXT GENERATION DATA INTEGRATION FLOWS

This section provides an overview of important characteristics for a next generation data integration flow. First, we describe an example scenario that will be used throughout the paper to illustrate various points; this scenario is depicted in Figure 3.

Consider a hypothetical, on-line, retail enterprise and its associated business process for accepting a customer order, fulfilling

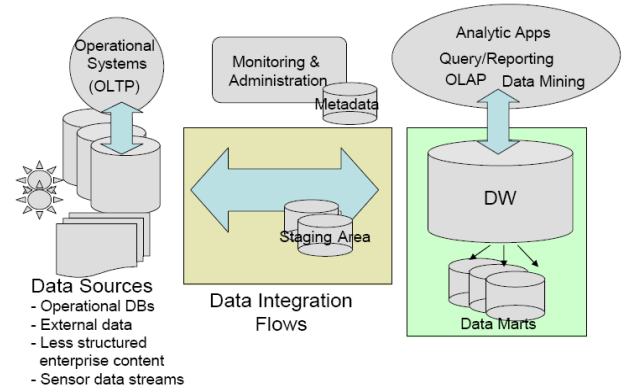


Figure 2. Next generation business intelligence architecture

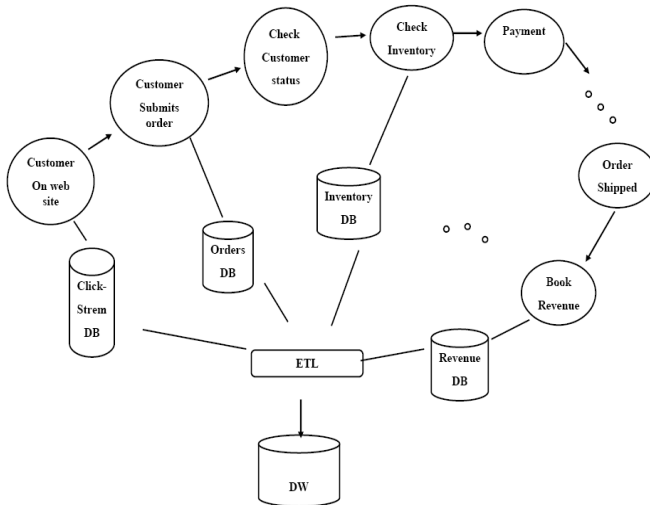
and shipping the order and booking the revenue. Such an *order-to-revenue* process involves a number of steps, utilizing various operational databases and an enterprise data warehouse. The process begins with a customer browsing the product web site and adding items to a shopping cart. During the session, some click-stream events may be captured in web logs or to a database.

Eventually, the customer proceeds to check-out which submits an entry to the order database. Then the customer status is checked to validate the order. This may involve interactions with an external agency for a credit check or for fraud detection. Next, the inventory database is checked to ensure the product is in stock. At this point, the order can be fulfilled so the customer payment is processed. This may involve additional steps (not shown). Finally, the order is shipped and the order revenue is added to the financial revenue database.

It is worth noting that the entire order-to-revenue process may last anywhere from seconds to days, depending on the customer behavior, external services, whether the items are in inventory, the desired shipping date, etc. We also note that the ETL may use an additional database of its own (not shown) as an intermediate staging area for processing. Finally, it is important to realize that the data warehouse is designed to answer specific business questions, such as the query in Figure 3. As the business changes and more demands are placed on the warehouse, the design of the integration flows may need to change.

ETL pipelines are responsible for extracting events and actions from the operational databases and loading them into the enterprise data warehouse. Today's enterprise data warehouses are dominated by structured data. In the future, we expect warehouses to incorporate new data types for semi-structured and unstructured data. For example, in our scenario, the click-stream database may include customer comments or feedback (in free text) on particular products or web-pages. Alternatively, some customer profile details (preferences, ratings) might be more easily stored as XML snippets rather than as relational tuples. Also for validation, the warehouse might include biometrics of a customer.

In current ETL solutions, the data flow is almost exclusively one-way from operational systems to the warehouse. In the future, we must support flows that are more general. For example, data cleansing, e.g., address disambiguation and customer deduplication, is an important function of ETL. The results of data



Example BI Query: what is our revenue this quarter as a function of customer traffic on our web site?

Figure 3. Example ETL scenario for an order-to-revenue business process

cleansing could be pushed back into the operational systems to improve their accuracy and reduce cleansing work. As another example, retail web-sites often use customer scores and profiles to make personalized offers. These profiles are computed from the warehouse and fed back to the operational systems.

In the future, we can expect operational BI users to demand lower latencies for time-sensitive data. For example, suppose our web-site displays an advertisement with a discount on a selected product. It is important to monitor the sales of that product to evaluate the utility of the ad. A nightly refresh cycle is too long. An effective campaign needs shorter cycles to provide fresher data, so that the campaign can be dynamically evaluated and adjusted. As another example, suppose we want to offer a discount to the user. We may want to base the offer on the user's up-to-the-minute profile (including the actions he has taken in this transaction), current inventory, and active marketing promotions, rather than on a historical customer segmentation model and last week's inventory. Again, this requires low latency in the ETL pipeline to capture and transport the user's click stream events and other information to the data warehouse within seconds so that his profile can be updated and the best offer computed on the fly.

Next-generation data integration flow solutions must support a systematic methodology for the entire lifecycle of flow design, implementation, and maintenance. In talking to practitioners, we uncovered a plethora of unmet needs. Today's ETL engines provide graphical interfaces for designing ETL flows, and scripting languages for implementing the designs. However, they do not capture or track business requirements, which are usually specified informally (in text documents), and practitioners have to translate these requirements into designs and implementations. This increases the cost of a project and results in a design that is hard to change. It also makes it difficult to determine how well a design meets the objectives or how a design change might affect a business objective.

The challenge is to link the business processes and objectives with the ETL design process. A data warehouse is, in effect, an encapsulation and abstraction of one or more business processes. Unfortunately, that connection to an actual business process is obscured or lost in a typical ETL project. For example, consider Figure 3. For the sake of discussion, we assume that the warehouse contains only booked revenue, i.e., orders that have shipped. The state of partial orders is held in the staging area. Depending on the refresh period, the data warehouse reflects past business activity that is days or weeks old. In addition, orders are processed at different rates so there is no guarantee that two orders submitted at the same time will appear in the warehouse at the same time. While the warehouse provides a consistent, historical view of completed orders, it does not reflect a complete view of the enterprise. For instance, there is no easy way to ask business questions such as "How many orders are currently in the state *Check-Customer-Status*?" Of course, that information is in the ETL staging area, but it's not in a form that is readily available to business managers or analysts. The goal should be to make ETL more cognizant of end-to-end business processes so as to enable a real-time dynamic view of an enterprise.

In addition to functional correctness (does the ETL pipeline correctly populate the data warehouse so the desired business views can be computed?), the design has to satisfy a number of other quality objectives. These include performance, reliability, maintainability, freshness, scalability, availability, flexibility, robustness, affordability, auditability, and traceability. In this paper, we refer to these collectively as QoX (to generalize terms such as Quality of Service, Quality of Data, Quality of Information, and so on). Practitioners have to produce ETL solutions that make tradeoffs among these QoX objectives to satisfy business objectives. The challenge is to capture the objectives formally, and to translate these into quality metrics that can then be used to evaluate design tradeoffs and create optimized implementations.

At the physical level, the next generation of data integration flow solutions should consider a larger number of implementation alternatives than they do today. Currently, the design choices are limited by the choice of implementation tool. Thus, for example, the decision to do an ETL design (where the transformations are performed in the staging area) versus an ELT design (where the data is loaded into the warehouse and the transformations are implemented using SQL operations in the data warehouse) is made very early in the project. In future, we expect to see more hybrids of these styles adopted as database engines become more scalable and parallel.

Other implementation choices have to be made. One is batch versus stream. Today's BI solutions typically run periodic batch ETL flows. However, to provide fresher data to the warehouse, it may be necessary to stream data from the sources as soon as they are updated. A second choice is pull versus push. In today's BI architecture, all data to be used for analysis and querying has first to be pushed to (i.e., consolidated in) the data warehouse. In the pull approach, the data integration flows are executed "on demand" when the data warehouse is queried. This is analogous to the federated database approach in which all data is left at the sources and queries are directed to the sources to extract and integrate data. (This approach has been the subject

of intense research for two decades [e.g., 10, 14].) While this approach can increase data freshness, and may be the only viable approach to dealing with some external data sources, adopting it in the BI context would require that all the data cleansing and transformation steps needed for reporting and analysis would have to be done at query processing time. Clearly, there are tradeoffs to be made. Additional alternatives include data partitioning, parallel processing, and so on.

Currently, to evaluate all such design alternatives, ETL practitioners use their personal judgment, or, at best, ad-hoc methodologies. Next generation solutions should adopt a systematic approach to design that balances the design tradeoffs against the project objectives.

3. LAYERED METHODOLOGY FOR THE DATA INTEGRATION FLOW LIFE-CYCLE

Every data integration project is unique. However, at a high level, each follows a four-phase pattern common to many types of services engagements. The *services lifecycle* begins with a pursuit phase which evaluates the feasibility of a project, i.e. determining the objectives, risks, costs, and benefits and ultimately making the go or no-go decision. Then comes a design phase which, given the project objectives and constraints, results in a detailed project plan that includes logical and physical models. The deployment (or implementation) phase then takes those models and produces executable artifacts, e.g., test suites, code for Ab Initio, DataState, Informatica or any other ETL tool. The final phase, management, executes the code, monitors performance and updates the models as necessary over the remaining lifetime of the project. Research on common information models for services engagements is highly relevant [21, 37].

In our view, next-generation data integration projects would benefit from a layered methodology for the lifecycle, which proceeds in successive, stepwise refinements from high-level business requirements, through several levels of more detailed specifications, down to execution models (Figure 4). Where possible, formal languages are used for the specifications and these specifications are influenced by key quality metrics. Specific optimizations are considered at each level of specification.

The first step of the methodology is the gathering of requirements and objectives. This is accomplished through interviews, examination of documents, and analysis of systems and data. The outcome of this step is the identification of business information objects, specifications and objectives for the project including the identification at this stage of key QoX metrics.

In our sample scenario of Figure 3, the information objects might include orders, products, customers, stores, suppliers and the associated relationships among these objects. The business specifications might include customer arrivals rates, order rates, product hierarchies, and so on. The specifications would also include the intended use cases for the data warehouse, e.g., types of reports or types of ad-hoc queries. Most important is that the specifications include the business processes that operate on the information objects. The objectives would include some QoX metrics and where possible, an initial attempt to quantify them. For example, maintainability might be specified

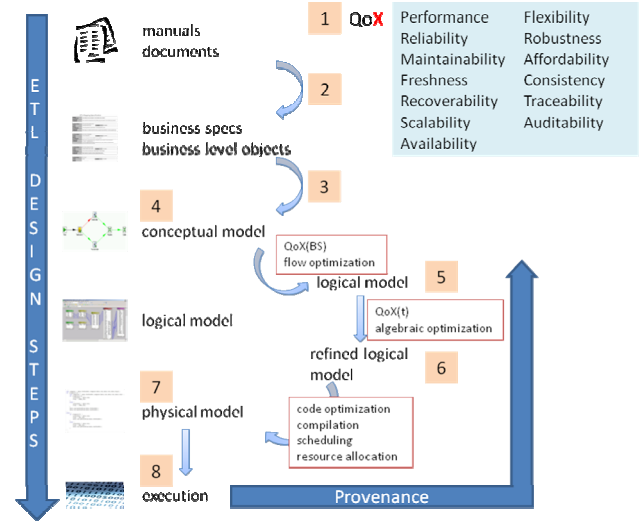


Figure 4. Layered, comprehensive approach for the integration flow lifecycle

as an important design goal with impact on the management phase of the project. But, it would be difficult to quantify. However, the required freshness of data might be known at this point. Some availability and performance metrics might be known, and so on.

A research challenge at this level is the expression of the business information objects and processes in terms of a formal model. This will establish a correspondence between business views at the business process level and operational views at the logical and physical level. One possibility for this formal model is to leverage languages developed for business processes [15, 38]. This might enable the use of design tools built around these languages.

The requirements gathering step synthesizes information from many disparate sources. Consequently, it is important that the resulting business specifications include provenance information. Since these specifications are ultimately used to derive the implementation, the provenance information is needed to track back a project detail to its ultimate source. Ideally, the provenance information would include some quality or accuracy measure, e.g., to identify the most authoritative source in the case of conflicting information.

Given this business level model, the next step is creation of a conceptual model. The conceptual model describes the data flows at a high level, the sources, the targets and the required mappings between them. It also describes dependencies and constraints among the flows. In our example scenario, there might be four flows, one for each of the operational databases that feed the warehouse. We note that existing ETL engines provide little or no support for the conceptual level. In a typical ETL project, the conceptual model would be expressed informally with using spreadsheets or annotated diagrams.

Conceptual modeling formalisms have been proposed [22, 34]. This enables the expression of dependencies among flows and certain types of optimizations. Certain flow-based optimizations are achievable based on the conceptual model. Flows with no intersection in their sources and targets could be run in parallel.

Flows with common sources might be considered for joint extraction.

Given a conceptual model, the next step is to generate a high-level logical model based on expanding the conceptual model to include specific transformations for the mappings. Typically, a logical model is expressed as a graph of algebraic operators with tuple (data) flow between operators. Some algebraic optimizations are best accomplished using this high-level logical model, such as optimizations that are independent of the physical implementation. The generation of the logical model should also take into account QoX metrics, and tradeoffs among them. For example, fast recoverability might require the inclusion of additional flows to establish landing tables that can be used to restart a flow. Given the initial high-level, logical model, cost-based techniques such as those in [27] can be used to produce a more detailed logical model matched to the physical implementation.

Finally, a physical model is generated, and this includes executable code. The physical model is dependent on the specific implementation technology chosen for implementing the data integration flows, e.g., custom scripts and ETL engine. Thus, a physical model for Informatica would differ from one for DataStage given the different capabilities of these ETL engines. As with the higher levels, the generation of the physical model and the optimization of the model are driven by the QoX metrics.

Some work has been done on automatic generation of ETL from conceptual or logical models, but only for a subset of ETL transformations that correspond to schema matching and mapping [13]. A semi-automatic method based on Semantic Web ontologies was described in [30]. However, a systematic approach that is based on QoX tradeoffs and successive refinement from the business level through to the physical implementation level is still lacking.

4. QoX METRICS AND TRADEOFFS

The software engineering community has proposed a set of measures for evaluating the quality of software designs. These could be adapted to evaluating the quality of data integration flow designs [35]. Exploiting the fact that such flows can be conveniently represented as graphs, these metrics are either simple graph properties (e.g., size of the graph, length of the longest path) or somewhat more complex quantities that require richer semantics and deeper understanding of the flow. Some examples of the latter are: *Modularity* or cohesion (of a module or transformation) refers to the extent to which a module or transformation performs exactly one job; *Coupling* (of a module or transformation) represents the number of inter-relationships between different modules or transformations; *Complexity* (of a module or transformation) refers to the number of inter-relationships among the components of a module or transformation. However, these metrics are rather abstract, and we decided to interview practitioners who specialize in ETL design and implementation to learn what they consider important.

We learnt that ETL designers have to deal with a host of quality objectives, which we refer to as QoX. While most of these qualities have been discussed in the software engineering literature, their definition and usage vary according to the specific domain. Below, we adapt these descriptions to the context of integration flows.

- *Reliability*. The probability that the ETL process will perform its intended operation during a specified time period under given conditions. Any reason for not performing the intended operation is considered to be a failure.
- *Maintainability*. The ability of an ETL process to be operated at the design cost and in accordance with service level agreements.
- *Freshness*. The ability of the system to provide the desired latency in updating the data warehouse. (Note that different data objects may have different freshness requirements.)
- *Recoverability*. The ability to restore an ETL process to the point at which a failure occurred within a specified time window.
- *Scalability*. The ability of an ETL process to handle higher volumes of data.
- *Availability*. The probability that the ETL process is operational during a specific time period, i.e., that the allocated physical resources of the system (e.g., processors, memory, external storage) will be available when needed. From the end user perspective, availability refers to the ability of the ETL process to provide the required data in the data warehouse within specified time and accuracy constraints.
- *Flexibility*. The ability to accommodate previously unknown, new or changing requirements.
- *Robustness*. The ability of an ETL process to continue operating well or with minimal harm, despite abnormalities (sometimes unpredictable abnormalities) in input, calculations, functionality, and so on, which stress the design assumptions.
- *Affordability*. The ability to maintain or scale the cost of an ETL process appropriately.
- *Consistency*. The extent to which the data populating the data warehouse is correct (i.e., satisfies integrity constraints) and complete.
- *Traceability*. The ability of an ETL process to track the lineage (provenance) of data and data changes.
- *Auditability*. The ability of an ETL process to protect data privacy and security, and to provide data and business rule transparency (usually for legal compliance purposes).

These descriptions are still quite informal and ETL practitioners today have to manually incorporate them into their designs in an *ad hoc* manner, based on their own experience and skills. One important challenge for future research is to define these metrics precisely and understand how to measure them. Some of the metrics are quantitative (for instance, reliability may be defined in terms of MTBF, the mean time between failures; recoverability may be measured in terms of MTTR, mean time to repair; freshness may be measured in time units; affordability may be measured in cost). Other metrics (for instance, maintainability, flexibility) may be more difficult to quantify.

The metrics may come into play at different levels of the methodology we described in Section 3. For example, freshness and reliability can be evaluated at the physical level, while their

implication at the conceptual or logical levels is not clear. On the other hand, maintainability and robustness can drive conceptual and logical modeling. Scalability and performance span the conceptual, logical, and physical levels.

A major challenge is to identify the interrelationships and dependencies among the metrics that lead to tradeoffs for alternative optimizations of data integration flows. For example, a design may sacrifice performance for maintainability. Also, partitioning and parallelization increase freshness, but hurt maintainability and robustness.

As a more complex example, consider tradeoffs among performance, freshness, reliability, auditability, recoverability, and cost. Consider an ETL flow based on Figure 3. Assume that we want to populate the data warehouse table DW_ORDERS with data coming from the source table S_ORDERS. Assume also that we need to consider only orders that have been shipped and placed in three states AZ, CA, and NV. Then, we want to group and aggregate the data by state and by day. Finally, before the loading of the DW_ORDERS, we have to replace the production keys with surrogate keys. This ETL flow is depicted in Figure 5a.

Let us examine different design alternatives, assuming that we can afford to use three machines to execute the ETL flow. The obvious objective is to improve *performance*, but also we need to ensure recoverability and maintainability. For improving performance, a choice is to introduce parallelism. Without loss of generality, we consider that the volumes of orders placed in each state are similar. Then, we can create three different ETL flows, each responsible for processing one state's data. A possible design is depicted in Figure 5b. For recoverability of an ETL flow, a popular technique is to enrich the flow with a number of recovery points such as backup tables, landing tables, or files. When a failure occurs, the process continues from the latest recovery point, instead of starting again from scratch. For the original ETL flow, the points (1) and (2) are candidate places to add recovery points. However, for the parallel case, there are six candidate recovery points, two for each flow. Here, there is a clear tradeoff between performance and recoverability.

On the other hand, the use of recovery points hurts freshness. For avoiding that, a solution is to ensure robustness (hence reducing the need for recovery) by means of redundancy. Instead of using the three machines for parallel processing of subsets of the source data, we use them for three parallel, redundant executions of the original flow (as in Figure 5c). Thus, even if a failure occurs in a flow, a voter placed at the end of the three branches can decide which copy to trust.

Now suppose that maintainability is also a QoX objective. Typical metrics that characterize the maintainability of a flow are its size, modularity, and complexity. Clearly, the maintainability of the flow depicted in Figure 5a is better than that of the flow depicted in Figure 5b. That is because the latter flow has larger size (more nodes), lower modularity (each conceptual task, e.g., surrogate keys assignment, is performed more than once), and greater complexity.

A systematic approach to design based on QoX tradeoffs benefits both the flow designer and the administrator. An interesting challenge is to devise a method for enabling comparison and

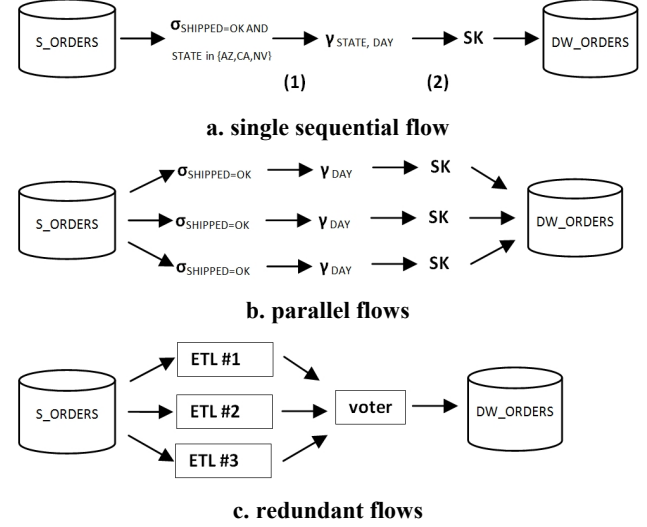


Figure 5. Examples of QoX tradeoffs

trade-offs of the different metrics. We adopt the NFR-Framework for making explicit the relationship between quality requirements and design decisions [7]. This framework distinguishes two types of requirements: functional (FR) and non-functional (NFR). The FR state specific functionalities of the system – intuitively, *what* the system must do; e.g., “The ETL process populates the Data Warehouse.” The NFR are attributes a system must have – intuitively, *how* the system must accomplish the *what*; e.g., “The ETL process populates the Data Warehouse *fast*.” Here, we focus on the NFR, and we discriminate two classes of metrics: the *qualitative* vs. the *quantitative*. The former contains “high level” QoX metrics that can be seen as *soft-goals*; e.g., “The ETL process should be *reliable*.” The latter contains “low-level” metrics that are functional parameters of the system; e.g., time window, execution time, recoverability time, arrival time, number of failures, latency of data updates, memory, space, CPU utilization uptime, throughput, number of processors, and so on. For example, the notion of “*reliable*” used in the above example can be clarified as: “the *mean time between failures (MTBF)* should be greater than X hours”. Another example could be “the *uptime* should be more than Y hours.”

A soft-goal interdependency graph is used to support the systematic modeling of the design [7]. Figure 6 shows an interdependency graph for an integration flow design that should be reliable, maintainable, and efficient. These three NFR are soft-goals expressed in the form of type[topic]. As these high-level requirements may denote different concepts to different people, their meanings should be defined. This is realized through a soft-goal refinement process, where soft-sub-goals are based either on topic (e.g., ETL System → {Software,Hardware}) or on type (e.g., Performance → {Time Performance, Space Performance}). The soft-goal interdependency graph shows the relationships among the soft-goals and the quantitative measures. Hence, Figure 6 illustrates that the degree of parallelism contributes extremely positively (++) to the fulfillment of the reliability[software] soft-goal, since it can be seen as a form of redundancy, which is a popular method for decreasing the probability of a failure. In contrast, parallelism affects negatively (-)

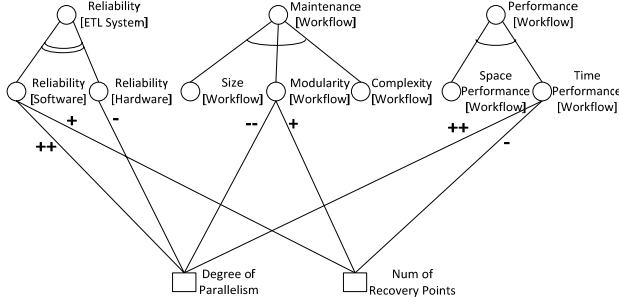


Figure 6. Example soft-goal interdependency graph

the reliability of hardware (more devices increase the probability of failure), extremely negatively (--) the modularity (each specific module of the system performs more than one task), but extremely positively (++) the time performance (the processes are executed faster). Similarly, the number of recovery points affects soft-goals differently.

However, if we analyze further the degree of parallelism and check how a design alternative such as partitioning, would affect the design, the result may be quite different. As we discussed in the example of Figure 5, if we partition the data into groups containing smaller but different volumes of data, that would be extremely good for time performance, but it does not really improve the reliability (since, if one of the parallel branches fails, then the final result would not be correct). If we choose a copy-partition, in which there are redundant parallel flows, then this would be extremely positive for the reliability of the system, but it would hurt time performance.

To summarize, an optimal design involves some degree of subjectivity, but it should satisfy the business objectives. The QoX metrics framework and techniques like the soft-goal interdependency graph may help toward the understanding and visualization of the design requirements. Simple measurements that quantify the positive or negative relationships among the soft-goals can give an intuitive perception of the design alternatives. We believe this is a fruitful area of research.

5. QoX-DRIVEN OPTIMIZATION

The layered approach to ETL design that was introduced in Figure 4 presents opportunities for optimization at each successive level of specification.

Suppose we are able to express the conceptual model in some formal language. Then, we can imagine an automated or semi-automated process that takes the conceptual model, along with supplementary information, such as the possible operators, and generates a logical model that specifies transformations for the flows. There may be several alternative *translations* from conceptual model to logical model, which can lead to different designs. The translation from the logical model to the physical model enables additional types of optimizations. The logical model, being expressed as a graph of operators, lends itself to algebraic optimization. Flow restructuring optimizations may also be applied at this level. The language of the physical model enables code optimizations that are dependent on the specific implementation technology in use (e.g., ETL engine, SQL code, and so on). However, we may also employ flow and algebraic optimizations at this level.

Simple operations:	DW operations:	System operations:
filter	SuperKey assignment	socket reader
join	SCD-1/2/3	socket writer
union	row (de-)normalize	file reader
sort	pivoting	file writer
group by	Flow operations:	stream lookup
diff	splitter	
function application	duplicator	Transfer operations:
Check operations:	merger	(de-)compress
key violation	Scripting operations:	encrypt/decrypt
null values	execute SQL script	file transfer
unique values	execute Java/C++ script	

Table 1. Representative ETL operations [3]

We believe that optimizations at all design levels should be driven by QoX metrics. These metrics, in effect, prune the search space of all possible designs, much like cost-estimates are used to bound the search space in cost-based query optimization.

5.1 Optimization Techniques and Challenges

The optimization of integration flows can exploit three classes of technique: *algebraic rewriting*, *flow restructuring*, and *adaptation to real-time environments*.

Algebraic rewriting. Since a data integration flow is expressed as a graph of operators, techniques analogous to database query optimization can be used. As in query optimization, flow optimization consists of selecting the execution order of the operations constituting the flow; and selecting the implementation method for each of these operations (e.g., join algorithms).

However, there are significant differences between optimizing integration flows and optimizing SQL queries, which render the former problem challenging. First, integration flows use a richer set of operators than the traditional relational operators. These include schema transformation routines (e.g., pivot/unpivot), data cleansing routines (e.g., de-duplication), transfer routines (e.g., ftp transfer), calls to external procedures (e.g., dll's), system calls, and so on. Table 1 presents a few representative operations supported by current ETL tools. Second, flows involve constraints on execution order, and in that sense, flow optimization is more akin to transaction or program optimization than query optimization. Third, a typical data integration design may consist of several flows. There are opportunities for sharing computations among these flows. This is analogous to the problem of multi-query optimization [8, 26, 28], with the caveat again that we have to deal with a richer set of operations and the execution ordering constraints. Fourth, database query optimization so far has been focused on performance (total query execution cost or response time), not on the more general QoX tradeoffs.

Despite the significance of optimization, so far the problem has not been extensively considered in the research literature on ETL. The existing studies mainly focus on a black-box optimization approach at the logical level, and concern the order with which the activities are placed in the flow [29]. More general rewriting that takes into account the semantics of the flow operations has not yet been addressed. A major inhibitor to progress

here is the lack of a formal language at the logical level (analogous to the relational algebra).

Commercial ETL tools provide little support for optimization. One technique that seems to give some promising results on the execution of a part of an ETL flow is the PushDown optimization supported by Informatica's PowerCenter [16, 17]. Push-Down is based on a two-pass processing. During the first pass, it starts from a source data store and checks if the subsequent transformations can be expressed in pure SQL. Then, it groups these transformations together with equivalent SQL expressions and executes the result in the source DBMS; this is similar to the approach taken in federated database systems. During the second pass, it executes the same procedure starting from the target data stores: the transformations that are placed at the end of the flow and can be expressed in pure SQL, are replaced by equivalent SQL expressions and the result is executed in the target DBMS (i.e., the data warehouse); this is similar to the ELT approach. The remaining transformations are executed in the data integration server; i.e., the ETL engine. If an operator that cannot be expressed in SQL is placed either early or late in the flow, this technique doesn't work. Furthermore, it applies to only part of the flow. The challenge of optimizing the entire flow remains.

Flow restructuring. Data integration flows are similar to business processes and can be expressed in process modeling languages (such as BPEL, process algebras, or process logics). Consequently, process optimization techniques such as parallelization, collapsing of long sequential chains, elimination of conditional branching nodes, etc., can be applied to flow optimization.

Two parallelization techniques, *pipelining* and *partitioning*, can be considered. For relatively small data volumes, the integration process can be divided into three pipelined sub-processes for extraction, transformation, and loading [33]. Most ETL tools work that way; however, pipeline can be delayed by blocking operations. As far as we are aware, no general solutions have been provided so far for avoiding blocking. For larger data volumes, partitioning is usually beneficial, and ETL tools support round robin, hash based, key range, pass-through, random, and follow-the-database partitioning. However, partitioning comes with the cost of merging the parallel flows.

Thus, there are some interesting research problems in flow restructuring: (a) how to automatically restructure an integration flow for pipelining and/or partitioning; (b) how to deal with blocking operations; (c) what are good candidate points for partitioning; (d) which parts of a flow should be parallelized; (e) what are good points for performing split or merge operations; and (f) how to automatically determine an appropriate partitioning technique.

Additional promising flow optimization techniques include a combination of SQL and MapReduce functions for distributed computation of large-scale cleansing transformations, sort operations, and the like [9]. This approach also seems to be promising for the integration of unstructured data as well, since MapReduce functions can be defined for text analysis, search, etc. Some preliminary ideas on optimizing integration operations implemented as SQL user-defined functions are reported in [5].

Adaptation to real-time environments. Although the requirement for freshness (low latency) does not impact the conceptual level of integration flows, at the physical level several challenges emerge. The three generic phases of an integration process, extract, transform, and load, have to deal with streaming data.

The enhancements needed to the extraction phase are not straightforward. The operational systems are tuned for transaction processing and cannot support the additional load of external applications pulling data from their databases at unknown times and in an *ad hoc* manner. Apart from the additional overhead, this requires having appropriate access privileges on the source data stores. Additionally, when the integration tool is allowed to interfere with the source systems, the source administrators have the obligation for propagating changes of their systems to the integration tool attached to their systems. For near real-time integration, extraction techniques include the use of messages, queues, web services, change data capture (CDC), and extraction in micro-batches.

In the transformation phase, the operations have to handle streaming data efficiently. There has been a lot of research on streaming select-project-join-aggregate queries. However, since integration flows involve a richer class of operators, a fresh look at this problem is needed. As a first step toward this task, the MeshJoin operator solves the problem of joining a data stream with a persistent relation. This is the core operation in transformations like the assignment of surrogate keys, the identification of the newly inserted/deleted/updated tuples, lookup operations, and so on [25].

Conceptually, loading seems to be the simplest phase of integration. In practice, however, this task is far from trivial. In general, DBMS engines are optimized for answering queries efficiently, not for enabling efficient loading of data, especially the construction of indexes and materialized views. Most commercial RDBMSs do provide external utilities for bulk loading (e.g. Oracle's sqldr, Neoview's Transporter). In the near real-time context, these approaches have to become resilient to the heavy bursts that characterize streaming data. Commercially, Teradata supports trickle loads. An example of a research effort in this direction is the RiTE approach, which considers real-time loading using micro-batches [24].

5.2 Implementation Styles

In addition to the optimizations described above, at the physical level, there are also the different implementation styles we mentioned in Section 2. While most data warehousing projects use ETL tools, some practitioners argue that ETL can be replaced by ELT (Extract-Load-Transform) or ETLT (Extract-Transform-Load-Transform). In the ELT style, after the extraction phase, data is loaded directly into the data warehouse server and all the transformations are executed there. In the ETLT case, the idea is to split the transformation phase into two groups of transformations, the first to be executed immediately after the extraction, and the second to be executed after a loading phase. The main argument for these alternatives is that the data warehouse servers are usually scalable, highly parallel machines and in principle could be better at optimizing transformations. The popularity of data warehouse appliances, which are very fast at loading and performing initial transformations, especially those that scan entire tables, has also contributed to

this trend. However, there are still many challenges in optimizing ELT flows, particularly since cleansing transformations can involve very complex SQL, including user-defined functions, which are not currently handled well by DBMS engines.

Another set of implementation choices is around the traditional push designs versus pull designs. In the pull design, data is left in the source data stores, and when a specific application needs some data, it is extracted from the sources and transformed on the fly. This approach resembles the traditional federated approach, and in the BI context we refer to it as on-demand ETL (or more generally, on-demand integration flows). In this case, it is advisable to maintain the source data as clean as possible, in order to avoid cleansing procedures on the fly, for performance reasons and for avoiding the cleansing of the same data more than once.

Between the two extremes of pull and push designs, another design alternative suggests populating the data warehouse only with a subset of data that are needed for applications with extreme response time requirements and for which freshness is not critical. Other data may be cached in the form of materialized views placed in the data staging area. These views enable tradeoffs between freshness and performance (and also benefit other QoX metrics since they provide recovery and lineage tracking points).

Tuning each of these alternatives is a challenge by itself. However, we believe the main challenge is to consider all of these implementation alternatives in order to do QoX-driven end-to-end optimization of data integration flows.

6. INTEGRATING ADDITIONAL DATA TYPES

The focus of data warehousing, and in particular of ETL, so far has been on structured data extracted from OLTP databases. Increasingly, enterprises have come to realize that for business intelligence purposes they need also to extract information from their unstructured and semi-structured data sources as well. By some accounts, over 80% of an enterprise's information assets are unstructured, mainly in the form of text documents (e.g., contracts, warranties, forms, medical reports, insurance claims, policies, reports, and customer support cases) and other data types such as images, video, audio, and other different forms of spatial and temporal data. For illustration purposes, consider our example from Figure 3. To construct an accurate user profile, it may be important to consider not just the transactional data in the various operational databases involved in the order-to-revenue process, but also to look at the content of web pages the user has browsed, his reviews or blog postings, and any contracts he may have with the enterprise. In general, unstructured data provides valuable contextual information for more informed operational decision making.

Recently, there have been efforts to incorporate unstructured or semi-structured data into data warehouses. For instance, [23] describe research on spatial and temporal data warehousing. The core idea underlying DW2.0 is the integration of structured and unstructured data [20].

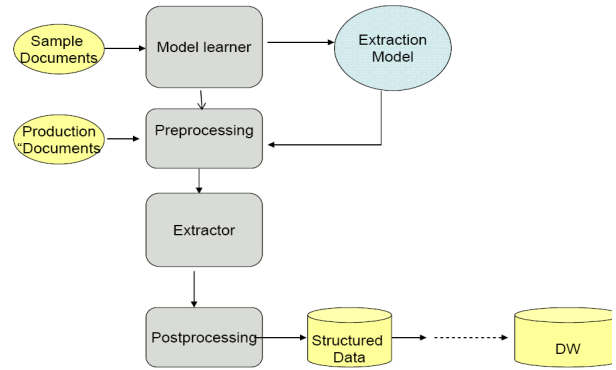


Figure 7. Information extraction pipeline for unstructured data

To incorporate unstructured data into the BI architecture of Figure 2, it is not sufficient to merely store the data in the warehouse. Rather, it is important to extract useful information from the unstructured data and turn it into structured data that can be stored, accessed and analyzed along with other structured data. However, this is not an easy task. Take, for example, the text in customer reviews, which are written in an ad-hoc manner. The lack of structure makes it hard to find the product and the features referred to in the review, and especially which features the customer likes and which he doesn't like.

Numerous *information extraction* techniques have been developed that try to learn models for the retrieval of relevant entities, relationships, facts, events, and so on, from text data [2]. Some of the most popular techniques are based on rule learning [31] and Hidden Markov Models [11]. Analogous techniques have been developed for extracting information from multimedia data such as text in video frames [1].

Figure 7 shows the typical pipeline for information extraction from text data sources, where the goal is to extract relevant data from a collection of documents; e.g., contract number, customer, and expiration date from contracts. Whatever the information extraction algorithm used, the source data always needs to be pre-processed to get rid of noise, transform it to the representation required by the method, etc. In addition, the output also needs to be post-processed to gather the structured data in the form of attribute-value pairs, which can then be transformed and loaded into the data warehouse. In the case of text, this pipeline of tasks corresponds to what has recently been named *textual-ETL* [19]. The pipeline involves numerous operations that are abstracted into high level modules in the figure. The extracted data from the unstructured sources often relates to data in structured sources so it is staged into a landing area and is then loaded into the data warehouse, where the information from both structured and unstructured sources is consolidated (e.g., contract data is related to customer data) for use by BI applications.

The challenge in textual-ETL consists in identifying how to abstract all the above tasks into operators that can be used to design, optimize and execute these flows in the same way as for structured data.

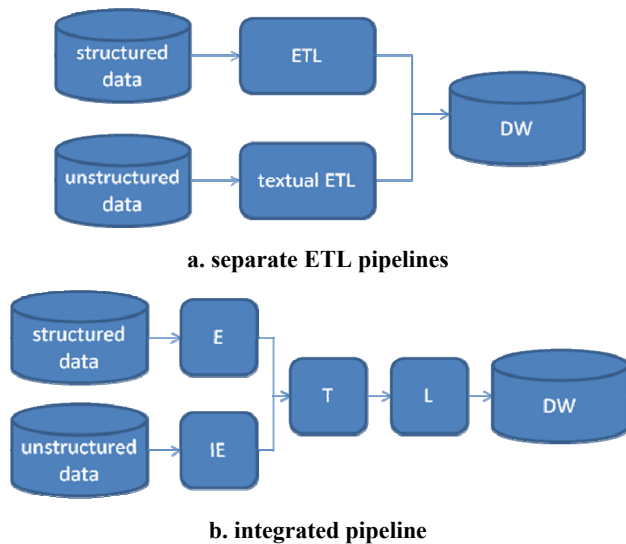


Figure 8. Data integration pipelines for structured and unstructured data

Using a uniform approach for ETL of structured and unstructured data makes it possible to use the same data warehouse infrastructure. In addition, by incorporating unstructured data into the same framework, QoX-driven optimization is enabled.

In fact, the same QoX metrics apply to unstructured data as well, although some of them may become more relevant in this context. For example, accuracy becomes critical for unstructured data given that the information is extracted (and turned into structured data) through learned extraction models that are hardly ever 100% accurate. Also, since information extraction algorithms are usually slow, tradeoffs between accuracy and performance may be important.

In addition, there are other design alternatives to be considered. Should there be separate pipes for the structured and unstructured data in the integration flows or only one? Having separate pipes (as shown in Figure 8a) is conceptually simpler to design. However, having a single pipe with two separate extraction stages but a single integrated transformation-load stage (as shown in Figure 8b) creates better opportunities for end-to-end optimization by bringing unstructured data into the same quality-driven design and optimization environment that has been described in previous sections.

7. CONCLUSION

Data integration flows are the back-end of a typical BI architecture. Today, the design and implementation of these flows is a labor-intensive activity, consuming a large fraction of the effort in data warehousing projects. We believe this is because the current generation of ETL tools provides little support for systematically capturing business requirements and translating these into optimized designs that meet the correctness and quality requirements. The next generation of BI solutions will impose even more challenging requirements (near real-time execution, integration of structured and unstructured data, and more flexible flow of data between the operational applications and analytic applications), resulting in even more complexity in integration flow design. Hence, it is important to create automated or

semi-automated techniques that will help practitioners to deal with this complexity.

In this paper, we have outlined the requirements we see for next generation data integration flows and the research challenges in meeting them. We have presented a layered methodology for starting with a business model of the enterprise and progressively refining the model to create conceptual, logical, and physical designs. We have introduced a suite of quality metrics we refer to collectively as QoX. We have described a framework that enables design tradeoffs to meet different quality requirements, and discussed various techniques for producing optimized designs. Finally, we have discussed the integration of additional data types. Only recently has the research community started to address these challenges, but many research problems remain.

8. REFERENCES

- [1] M. Anthimopoulos, B. Gatos, I. Pratikakis. Multiresolution text detection in video frames. In VISAPP (2), pp. 161-166, 2007.
- [2] M. Berry, M. Castellanos (Eds). Survey of Text Mining II: Clustering, Classification and Retrieval. Springer Verlag, 2008.
- [3] M. Castellanos, A. Simitsis, K. Wilkinson, U. Dayal. Automating the Loading of Business Process Warehouses. In EDBT, 2009.
- [4] S. Chaudhuri, U. Dayal, V. Ganti. Database Technology for Decision Support Systems. In IEEE Computer 34(12), pp. 48-55, December 2001.
- [5] Q. Chen, M. Hsu. Data Continuous SQL Process Model. In CoopIS, pp. 175-192, 2008.
- [6] S. Chen, L. Bao, P. Chen. OptBPEL: A Tool for Performance Optimization of BPEL Process. In Software Composition, pp. 141-148, 2008.
- [7] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishing, 1999.
- [8] N.N. Dalvi, S.K. Sanghai, P. Roy, S. Sudarshan. Pipelining in Multi-Query Optimization. In PODS, 2001.
- [9] J. Dean, S. Ghemawat. MapReduce. Simplified Data Processing on Large Clusters. In Sixth Symposium on Operating System Design and Implementation, 2004.
- [10] A. Elmagarmid, M. Rusinkiewicz, A. Sheth. Management of Heterogeneous and Autonomous Database Systems. Morgan Kaufmann, 1999.
- [11] Freitag, A. McCallum. Information Extraction with HMM Structures Learned by Stochastic Optimization. In National Conference on Artificial Intelligence, 2000.
- [12] P. Gillin. BI @ the Speed of Business. Computer World Technology Briefings. December 2007. Available at: http://resources.computerworld.com/sas_imw/registration.php?item=12&tab=1.
- [13] L.M. Haas, M.A. Hernández, H. Ho, L. Popa, M. Roth. Clio grows up: from research prototype to industrial tool. In SIGMOD, pp. 805-810, 2005.

- [14] A.Y.Halevy, A. Rajaraman, J.J. Ordille. Database Integration: The Teenage Years. In VLDB, pp, 9-16, 2006.
- [15] R. Hull. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In ODBASE Conference, pp. 1152-1163, 2008.
- [16] Informatica. Pushdown Optimization. Available at: http://www.informatica.com/INFA_Resources/ds_pushdown_optimization_6675.pdf
- [17] Informatica. How to Achieve Flexible, Cost-effective Scalability and Performance through Pushdown Processing. White paper, November 2007.
- [18] W.H. Inmon. Building the Data Warehouse. John Wiley, 1993.
- [19] W. H. Inmon, A. Nesavich. Tapping into Unstructured Data: Integrating Unstructured Data and Textual Analytics into Business Intelligence. Morgan Kaufmann, 2007.
- [20] W.H. Inmon, D. Strauss, G. Neuschloss. DW 2.0. The Architecture for the Next Generation of Data Warehousing. Morgan Kaufmann, 2008.
- [21] H.A. Kuno, K. Yuasa, K. Govindarajan, K. Smathers, B. Burg, P. Carau, K. Wilkinson. Governing the Contract Lifecycle: A Framework for Sequential Configuration of Loosely-Coupled Systems. In DNIS, pp. 264-279, 2005.
- [22] S. Luján-Mora, P. Vassiliadis, J. Trujillo. Data Mapping Diagrams for Data Warehouse Design with UML. In ER, pp. 191-204, 2004.
- [23] E. Malinowski, E. Zimanyi. Advanced Data Warehouse Design. From Conventional to Spatial and Temporal Applications. Springer, 2009.
- [24] C. Thomsen, T.B. Pedersen, W. Lehner. RiTE: Providing On-Demand Data for Right-Time Data Warehousing. In ICDE, pp. 456-465, 2008.
- [25] N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, N.-E. Frantzell. Supporting Streaming Updates in an Active Data Warehouse. In ICDE, pp. 476-485, 2007.
- [26] P. Roy, S. Seshadri, S. Sudarshan, S. Bhohe. Efficient and Extensible Algorithms for Multi Query Optimization. In SIGMOD, pp. 249-260, 2000.
- [27] T.K. Sellis, A. Simitsis. ETL Workflows: From Formal Specification to Optimization. In ADBIS, pp. 1-11, 2007.
- [28] T.K. Sellis. Multiple-Query Optimization. In ACM Trans. Database Syst. 13(1), pp. 23-52, 1988.
- [29] A. Simitsis, P. Vassiliadis, T.K. Sellis. Optimizing ETL Processes in Data Warehouses. In ICDE, 2005.
- [30] D. Skoutas, A. Simitsis. Designing ETL Processes Using Semantic Web Technologies. In DOLAP, pp. 67-74, 2006.
- [31] S. Soderland: Learning Information Extraction Rules for Semi-Structured and Free Text. In Machine Learning 34(1-3), pp. 233-272, 1999.
- [32] V. Tziouvara, P. Vassiliadis, A. Simitsis. Deciding the Physical Implementation of ETL Workflows. In DOLAP, pp. 49-56, 2007.
- [33] P. Vassiliadis, A. Simitsis. Near Real Time ETL. In Springer Annals of Information Systems, Vol. 3, pp. 19-29, 2008.
- [34] P. Vassiliadis, A. Simitsis, S. Skiadopoulos. Conceptual modeling for ETL processes. In DOLAP, pp. 14-21, 2002.
- [35] P. Vassiliadis, A. Simitsis, M. Terrovitis, S. Skiadopoulos. Blueprints and Measures for ETL Workflows. In ER, pp. 385-400, 2005.
- [36] C. White. The Next Generation of Business Intelligence: Operational BI. DM Review Magazine, May 2005
- [37] K. Wilkinson, H.A. Kuno, K. Govindarajan, K. Yuasa, K. Smathers, J. Nanda, U. Dayal. Enabling Outsourced Service Providers to Think Globally While Acting Locally. In EDBT, pp. 1106-1109, 2006.
- [38] WS-BPEL Version 2.0, Oasis. Available at: <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>